CS103L SPRING 2020

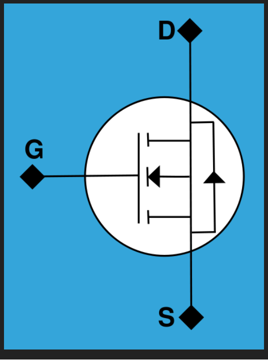
UNIT 1: TYPES, VARIABLES,EXPRESSIONS,C++ BASICS

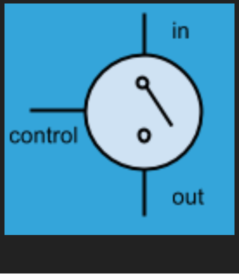
LEARNING OBJECTIVES

- Understand representations
- Understand types
- Understand basic pieces of C++ program
    - Statements, expressions, variables, function calls

WHY 0/1

- Digital computer memory holds binary numbers
  - Binary = two values
- Why?
  - Fundamental Unit = digital transistor = switch = on or off
  - 0 and 1 are (arbitrary, but mathematically convenient) values





KINDS OF INFORMATION

- Non-authoritative list
  - Numbers
  - Text
  - Sound
  - Images/Video

KINDS OF INFORMATION

- All very different
- Computer can only store 1/0's
- So we define a representation

REPRESENTATION

- Representation
  - Definition (or mapping) from digital data to values (actual information)

INTERPRET THIS

- 01000001
- 8-bit binary number.
- What does it mean?
- Representing an integer = 65 (base 10)
- Representing a real number = 8.5 (floating point system)
- Representing a character = ‘A’ (ASCII System)

REPRESENTATION (REVISITED)

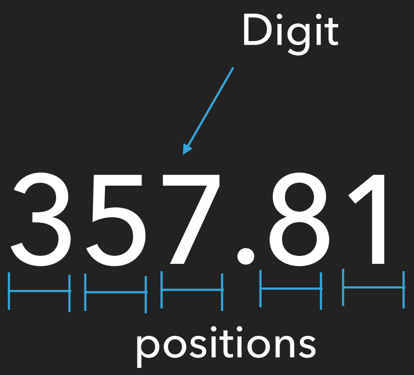
- ‘value’ (information) = bits (1/0’s) + representation

NUMBER THEORY BACKGROUND

- Humans use base 10
    - Why?

ANATOMY OF A BASE 10 NUMBER

- Each digit = place value
- Position = implied power of 10



Digit

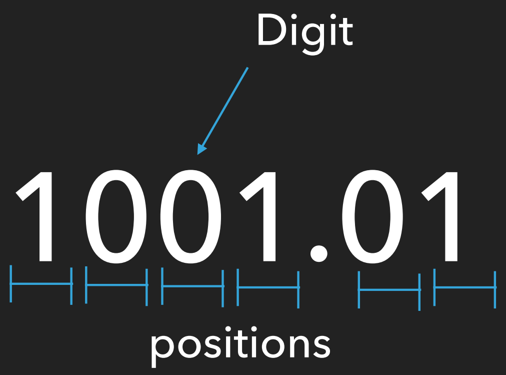
357.81

positions

$$\text{value} = 3*10^{2} + 5*10^{1} + 7*10^{0} + 8*10^{-1} + 1*10^{-2}$$

ANATOMY OF A BASE 2 NUMBER

- Each digit = place value
- Position = implied power of 2



Digit

1001.01

positions

$$\text{value} = 1*2^{3} + 0*2^{2} + 0*2^{1} +1*2^{0} + 0*2^{-1}+1*2^{-2}$$

REPRESENTATION SIZE

- How many things can a binary number represent?
  - How many unique states are there?
  - Example is usually integer numbers, but remember could be anything
- Given a *n* digit number of base *r*, how many unique things can be identified?
  - r^n

REPRESENTATION SIZE

- 2 digit base 10 numbers?

  ____ ____
  0-9 0-9

- 3-digit base 10?

  ____ ____ ____
  0-9 0-9 0-9

- 4-bit binary number?

  ____ ____ ____ ____
  0-1 0-1 0-1 0-1

- 6-bit binary number?

  ____ ____ ____ ____ ____ ____
  0-1 0-1 0-1 0-1 0-1 0-1

REPRESENTATION SIZE

- 2 digit base 10 numbers? ____ ____ Answer: 00-99 = 100
  0-9 0-9
- 3-digit base 10? ____ ____ ____ Answer: 000-999 = 1000
  0-9 0-9 0-9
- 4-bit binary number? ____ ____ ____ ____ Answer: 0000-1111 = 16
  0-1 0-1 0-1 0-1
- 6-bit binary number? ____ ____ ____ ____ ____ ____ A: 000000-111111 = 64
  0-1 0-1 0-1 0-1 0-1 0-1

POWERS OF TWO

- You should memorize these
- It’s super useful

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096

REPRESENTATIONS IN C++

- In C++ a representation is called a type
- Agreement between the programmer and compiler on what the binary numbers mean (the information)

REPRESENTATION #1: INTEGERS

- What is an integer?
- Properties?
    - Signed vs. unsigned

INTERGERS IN C++

- Two properties of integer types:
    - Width (number of bits)
    - Signed vs. unsigned

UNSIGNED INTEGERS

- Bits represent zero and positive integers

Width	Name	Unique Values	Range
8	unsigned char	256	0→255
16	unsigned short	65536	0→65535
32	unsigned int	2^{32} ~ 4B	0→ 2^{32}-1
64	unsigned long long	2^64 ~ $1.6x10^{19}$	0→ 2^{64}-1

SIGNED INTEGERS

- Bits represent negative and positive integers

Width	Name	Unique Values	Range
8	char	256	-128→127
16	short	65536	-32768→32767
32	int	2^{32} ~ 4B	-2B → +2B
64	long long	2^64 ~ 1.6x10^{19}	-8x10^{18}→ 8*10^{18}

COMPARING UNSIGNED VS. SIGNED

Width	Name	Unique Values	Range
8	unsigned char	256	0→255
16	unsigned short	65536	0→65535
32	unsigned int	2^{32} ~ 4B	0→ 2^{32}-1
64	unsigned long long	2^64 ~ 1.6x10^{19}	0→ 2^{64}-1

Width	Name	Unique Values	Range
8	char	256	-128→127
16	short	65536	-32768→32767
32	int	2^{32} ~ 4B	-2B → +2B
64	long long	2^64 ~ 1.6x10^{19}	-8x10^{18}→ 8*10^{18}

These three columns are the same

REPRESENTATION #2: FLOATING POINT

- What about “real” numbers (fractions)
- Think about scientific notation
  - 6.03 x 10^{23}
  - 6.6254 x 10^{-27}
- Decimal: ±D.DDD x $10^{\pm exp}$
- Binary: ±B.BBB x $2^{\pm exp}$

FLOATING POINT TYPES IN C++

- Bits represent a floating point number
  - Notice it might be an approximation to a “real-world” number

Name	Width	Range
float	32	±7 digits x $10^{\pm38}$
double	64	±16 digits x $10^{\pm308}$

REPRESENTATION ASIDE: HEXADECIMAL NOTATION

- Binary numbers get long fast:
  - 32-bits: 1110 1101 0101 0101 0111 0100 1010 1001
  - CS people came up with short-cut: hexadecimal notation
    - 16 symbols: 0 - F

1110	1101	0101	0101	0111	0100	1010	1001
E	D	5	5	7	4	A	9

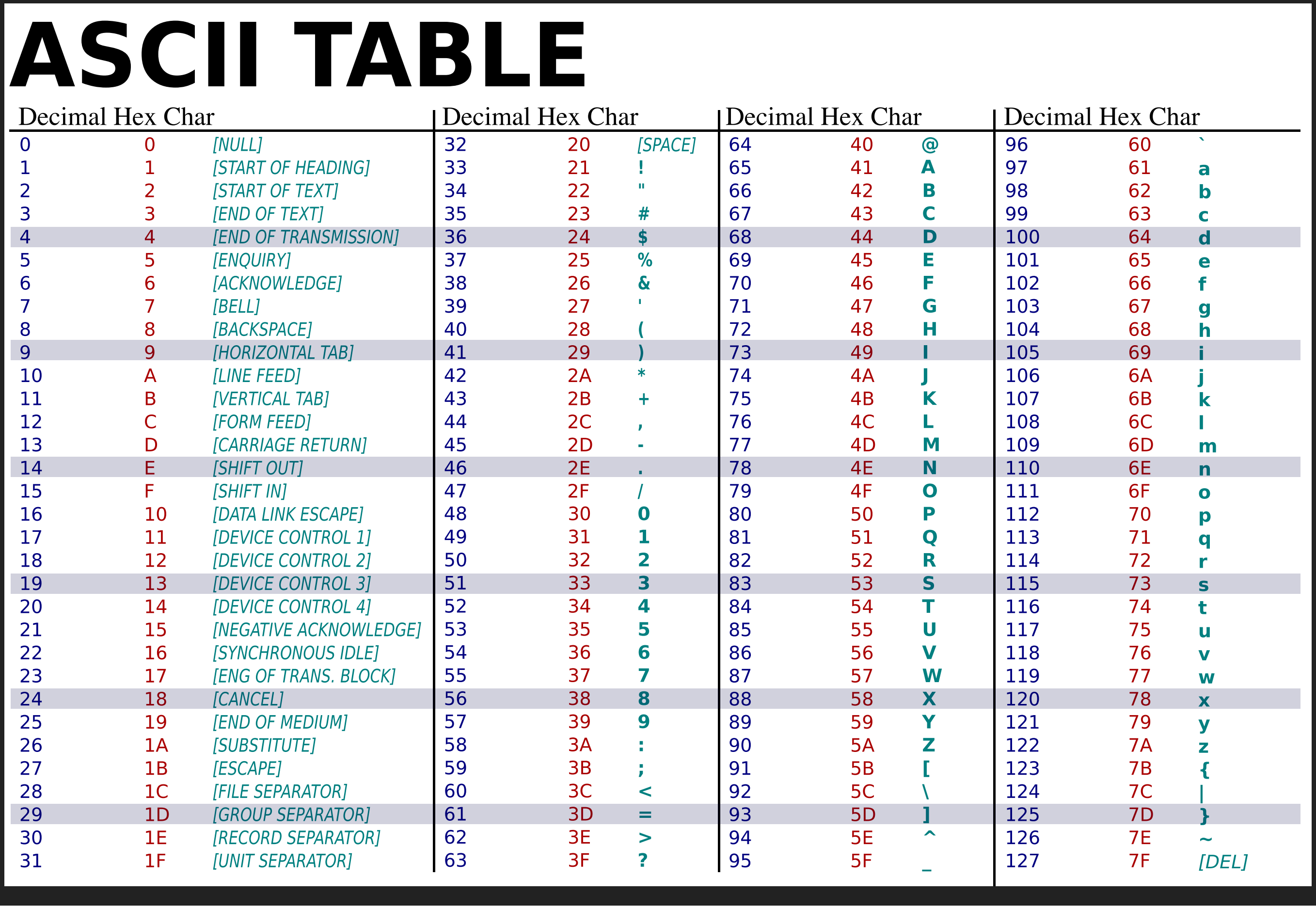
- Often grouped in pair: 0xED 0x55 0x74 0xA9
- Pair = 8 bits = 1 byte = smallest addressable memory size

Digit	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

NEXT TYPE: TEXT

- Bits represent text characters
- ASCII (defacto-standard)
- 8 bits
  - How many characters?
- Unicode (modern standard)
- 16-bits
  - How many characters?

ASCII TEXT REPRESENTATION



ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

ASCII IN C++

- Use ‘unsigned char’ or ‘char’ type to hold one character
- ‘H’ = 0x48 ‘e’ = 0x65 ‘l’ = 0x6C ‘l’ = 0x6C ‘o’ = 0x6F
- Strings = “Hello”
  - C-strings = arrays of chars
  - C++ strings = type (more on these later)
- cout << “Hello\n”;
  - prints out Hello and then a ‘new-line’ (moves cursor left and down)
- Other unprintables: tab ‘\t’

ASCII VS. UNICODE

- ASCII originally 7-bit: 0-9, A-Z,a-z + some other common characters
- Extended ASCII 8-bit: a few international characters
- Unicode: 16 bits, enough for most languages
- We won’t worry about Unicode in this class

TYPES REVIEW

- Everything in C++ has a type: int, char, double…
    - Amount of memory per one item of a particular type depends on the type
        - int = 32 bits = 4 bytes
        - double = 64 bits = 8 bytes
        - char = 8 bits = 1 byte

BASIC PIECES OF A C++ PROGRAM

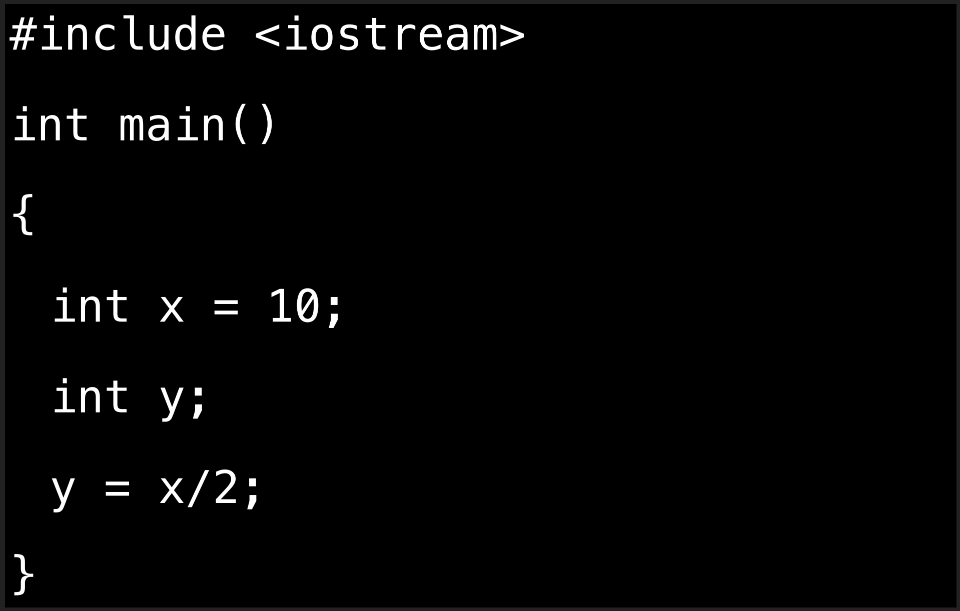
- Statements
- Constants
- Variables
- Expressions

STATEMENTS

- Essentially the basic building block.
- Tells the compiler one thing to do:
  - Declare a variable
  - Do some math
  - Move some data

STATEMENTS

- End in a semi-colon
- Example:
  - this program has 3 statements
    - actually 4…



```
#include <iostream>
int main()
{
  int x = 10;
  int y;
  y = x/2;
}
```

CONSTANTS

- Things (numbers, strings, etc.) that you put in your code
- Have types
  - integers, floating point, characters
- Example
- Usually used to initialize variables

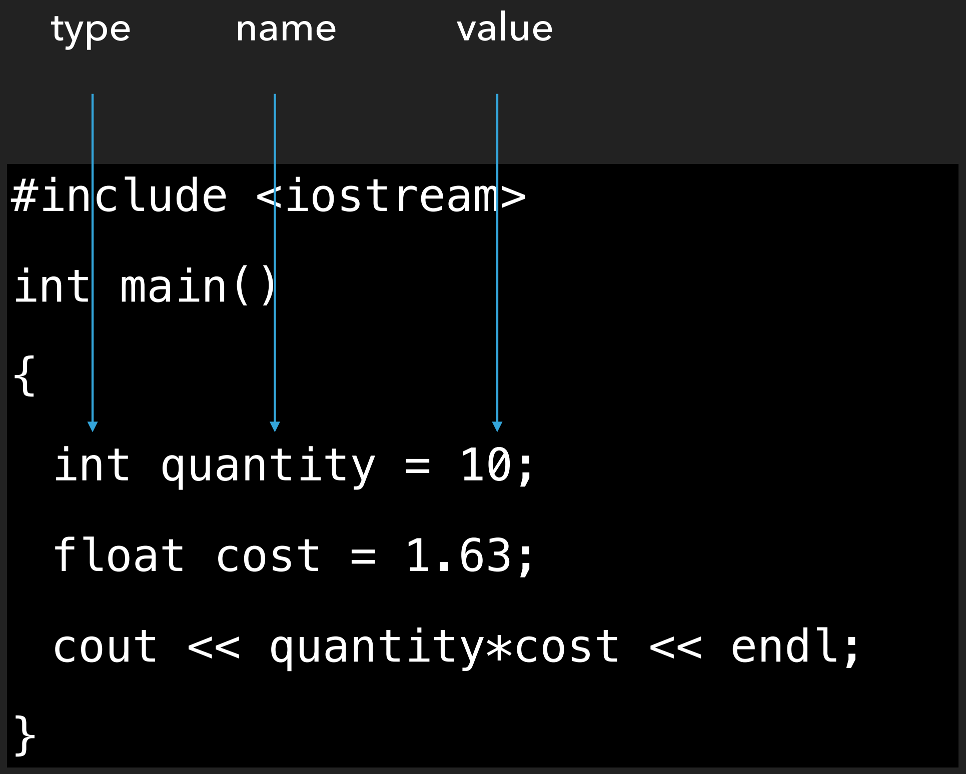
```
#include <iostream>
int main()
{
  int x = 10;
  float y = 12.5F;
  char *str = “Hello!”;
  bool cond = true; //also false
}
```

VARIABLES

- A program needs to operate on data (information) to do it’s job
- Us humans need easy ways to refer to/identify/remember what something is
- We create variables (of a particular type) to hold information
- We give them names
  - x, i, first_name, high_score
- *PROGRAMMER* decides what variables are needed to solve the problem
  - Think about our recipe
- Compiler sets aside the right amount of memory for you, lets you use easy to remember name to refer back to the information

C++ VARIABLES

- C++ variables have
  - type and name (programmer chosen)
  - location (compiler chosen)
  - value (set by program operation)
  - Example with two variables



```
#include <iostream>
int main()
{
  int quantity = 10;
  float cost = 1.63;
  cout << quantity*cost << endl;
}
```

VARIABLE TIPS

- How to chose which variables you need?
- Choose good names (area, x_size, y_size, first_name)
    - Dictated by your solution (algorithm) to the problem
        - Values entered at run-time
        - Computed values: calculate once, use many times
            - Ex: need (3*x^2 + 4*x) several times in a program. Calculate once, assign to a variable
        - Desire to make code more read-able
            - Ex: calculating area of a rectangle. Length of one side = 3*x + y + 5*z, length of the other side = 72*k - 32*j
            - s1 = 3*x + y + 5*z; s2 = 72*k - 32*j; area = s1 * s2;
            - or area = (3*x + y + 5*z)*(72*k - 32*j);

VARIABLES NEEDED

- What variables might we need?
    - Calculator
    - TV
    - Tic-Tack-Toe

ARITHMETIC OPERATORS

- Now that we have variables (containing data), we need to compute with them

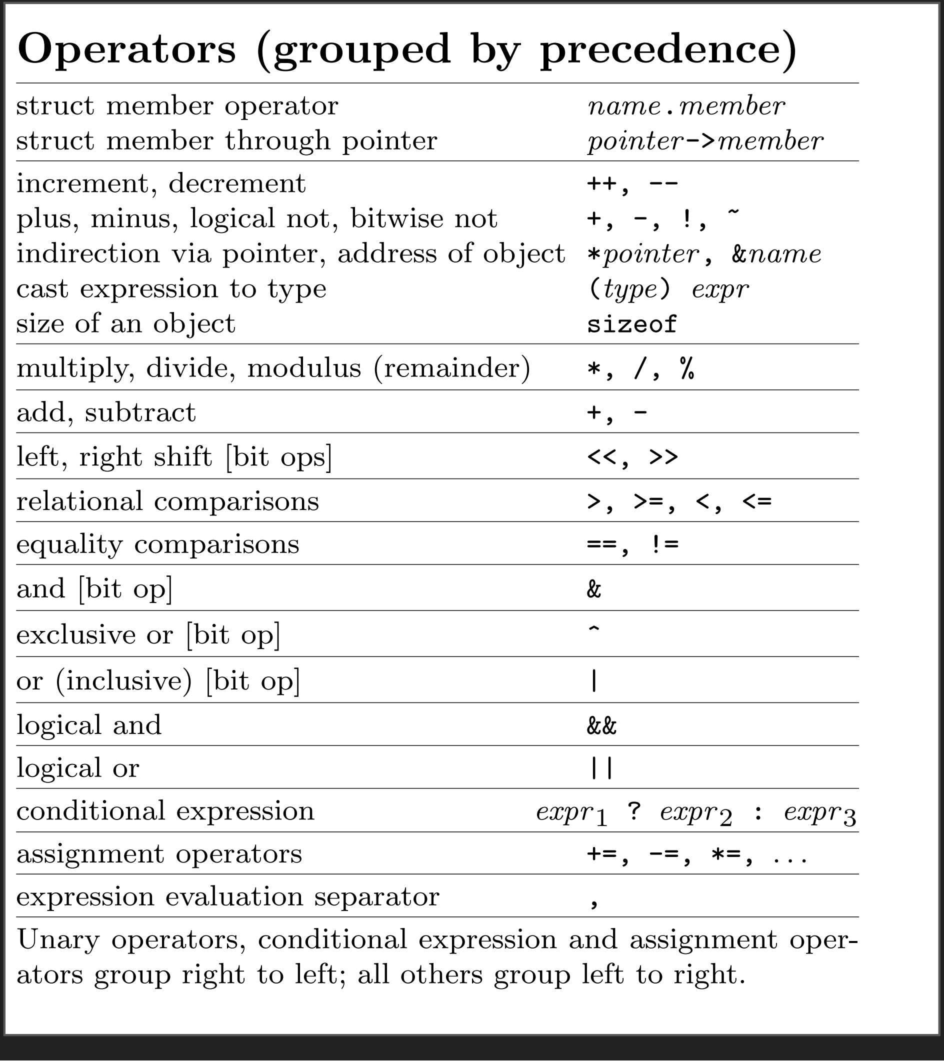
Operator	Name	Example
+	Addition	z = x + y + 5;
-	Subtraction	z = x - y;
*	Multiplication	z = x*y;
/	Division	int x = 10/3; //3 double x = 10.0/3; //3.33
%	Integer Modulus	z = 17 % 5; //2
++ or –	Increment or Decrement	x++; y–;

INTEGER VS. DOUBLE (FLOATING POINT) DIVISION

- If all operands are integer, compiler performs integer division
  - Examples:
    - 5/2 = 2;
    - 10/3 = 3;
    - 200/300 = 0;
- This can trip up even veteran programmers
  - More in a few slides…

OPERATOR PRECEDENCE

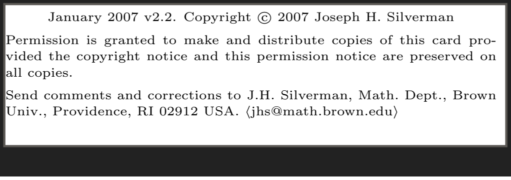
- Like PEMDAS we all learned in school
- Operators at top done first
  - Operators at same level usually evaluated left-to-right
- Ex: 2*-4-3+5/2;
- Programming tip:
  - Use parens to add clarity
    - (2*-4)-3+(5/2);



Operators (grouped by precedence)

struct member operator	*name*`.`*member*
struct member through pointer	*pointer*`->`*member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !, ~`
indirection via pointer, address of object	`*`*pointer*`, &`*name*
cast expression to type	`(`*type*`)` *expr*
size of an object	`sizeof`
multiply, divide, modulus (remainder)	`*, /, %`
add, subtract	`+, -`
left, right shift [bit ops]	`<<, >>`
relational comparisons	`>, >=, <, <=`
equality comparisons	`==, !=`
and [bit op]	`&`
exclusive or [bit op]	`^`
or (inclusive) [bit op]	`|`
logical and	`&&`
logical or	`||`
conditional expression	$expr_1$ `?` $expr_2$ `:` $expr_3$
assignment operators	`+=, -=, *=, ...`
expression evaluation separator	`,`

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.



OPERATOR PRACTICE

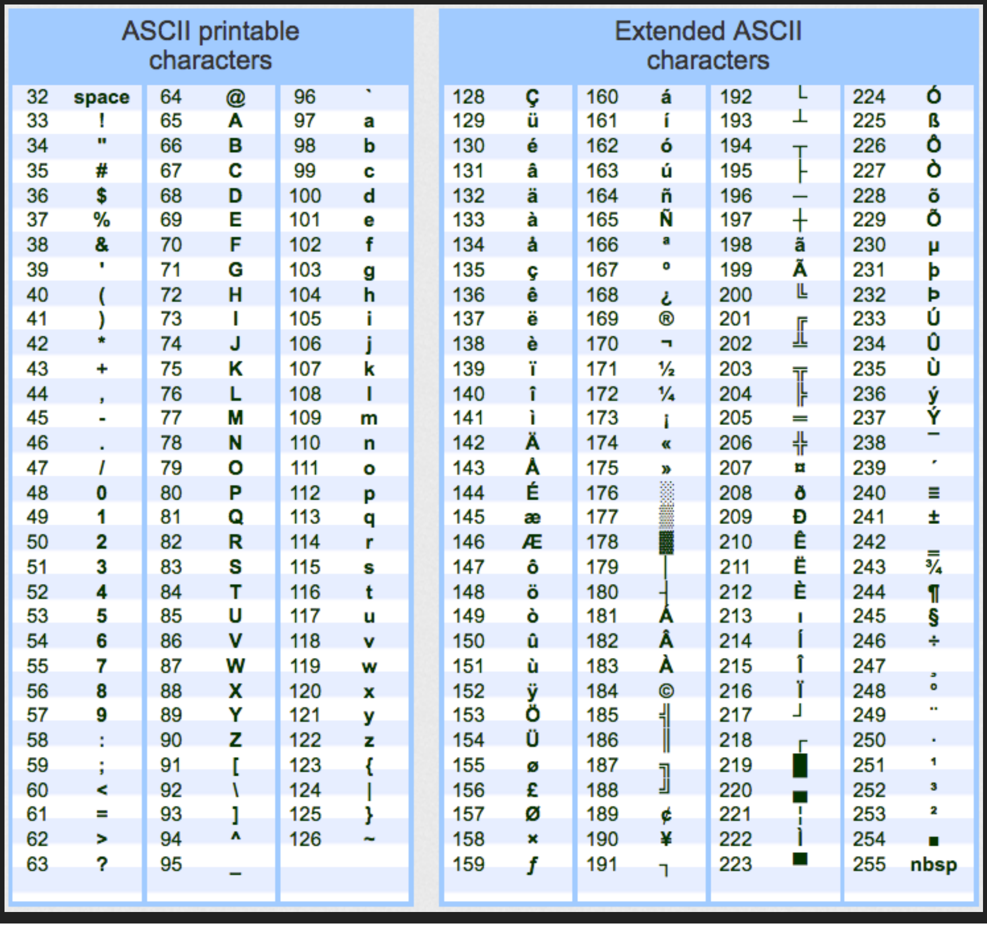
- D.S. Malik, C++ Programming, 5thEd., Ch. 2–Q6:
    - 25/3
    - 20-12/4*2
    - 33 % 7
    - 3 –5 % 7
    - 18.0 / 4
    - 28 -5 / 2.0
    - 17 + 5 % 2 -3

IN CLASS EXERCISES

- maxplus
- char_arith

CHARACTERS ARE NUMBERS – UNDERSTANDING ASCII

- Remember ASCII is a representation
    - Mapping from numbers to information
    - Information is characters
    - So we can do math with characters
        - ‘a’ + 1 = ‘b’
    - Weird, but helpful



ASCII printable characters						Extended ASCII characters							
32	space	64	@	96	`	128	Ç	160	á	192	└	224	Ó
33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
34	"	66	B	98	b	130	é	162	ó	194	┬	226	Ô
35	#	67	C	99	c	131	â	163	ú	195	├	227	Ò
36	$	68	D	100	d	132	ä	164	ñ	196	─	228	õ
37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	Õ
38	&	70	F	102	f	134	å	166	ª	198	ã	230	µ
39	'	71	G	103	g	135	ç	167	º	199	Ã	231	þ
40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Þ
41	)	73	I	105	i	137	ë	169	®	201	╔	233	Ú
42	*	74	J	106	j	138	è	170	¬	202	╩	234	Û
43	+	75	K	107	k	139	ï	171	½	203	╦	235	Ù
44	,	76	L	108	l	140	î	172	¼	204	╠	236	ý
45	-	77	M	109	m	141	ì	173	¡	205	═	237	Ý
46	.	78	N	110	n	142	Ä	174	«	206	╬	238	¯
47	/	79	O	111	o	143	Å	175	»	207	¤	239	´
48	0	80	P	112	p	144	É	176	░	208	ð	240	≡
49	1	81	Q	113	q	145	æ	177	▒	209	Đ	241	±
50	2	82	R	114	r	146	Æ	178	▓	210	Ê	242	‗
51	3	83	S	115	s	147	ô	179	│	211	Ë	243	¾
52	4	84	T	116	t	148	ö	180	┤	212	È	244	¶
53	5	85	U	117	u	149	ò	181	Á	213	ı	245	§
54	6	86	V	118	v	150	û	182	Â	214	Í	246	÷
55	7	87	W	119	w	151	ù	183	À	215	Î	247	¸
56	8	88	X	120	x	152	ÿ	184	©	216	Ï	248	°
57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	¨
58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
59	;	91	[	123	{	155	ø	187	╗	219	█	251	¹
60	<	92	\	124	|	156	£	188	╝	220	▄	252	³
61	=	93	]	125	}	157	Ø	189	¢	221	¦	253	²
62	>	94	^	126	~	158	×	190	¥	222	Ì	254	■
63	?	95	_			159	ƒ	191	┐	223	▀	255	nbsp

www.
theASCIIcode
.com.ar

COMPUTERS DO MATH, RIGHT?

- So, if computers do math…
    - What is 5 + 3/2 (as far as C++ is concerned)?

THE ANSWER IS 6.5?

- or is the answer 6?
- Computers love integer math - very fast
    - C/C++ defaults to integer math if the operands are integers
    - 5 + 3/2 = 6
- To get 6.5 we need to use casting

CASTING

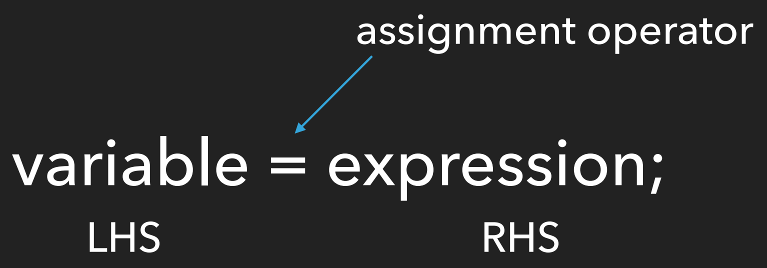
- Casting explicitly tells compiler how to treat a number (or variable)
- Three ways to get 6.5:
  - 5.0 + 3.0/2.0 (explicitly use doubles, or double typed variables)
  - 5 + 3/2.0 (implicit casting caused by a mixed type expression
    - known as promotion
  - (double)5 + (double)3/(double)2
    - Explicit casting syntax - look in operator table

EXPRESSIONS

- Expressions are pieces of C++ code that are evaluated to a result
  - Often the RHS of an assignment
- x + 1
- sin(x) + 2
- (x || y)

ASSIGNMENT OPERATOR

- Very commonly used operator, think like equals in math
- Used to assign values to variables



assignment operator

variable = expression;

LHS

RHS

- RHS = use these values and variables to calculate an answer
- LHS = where to put the answer
- Variable can be in LHS and RHS
  - uses current value to calculate expression, assigns (updates) back to the variable

SHORT CUT OPERATORS

- Every byte used to matter (when floppy disks were 1.4M)
  - Also, programmers are lazy
- x = x + 1; → x++;
- x = x/2; → x/=2;
- x = x*2; → x*=2;

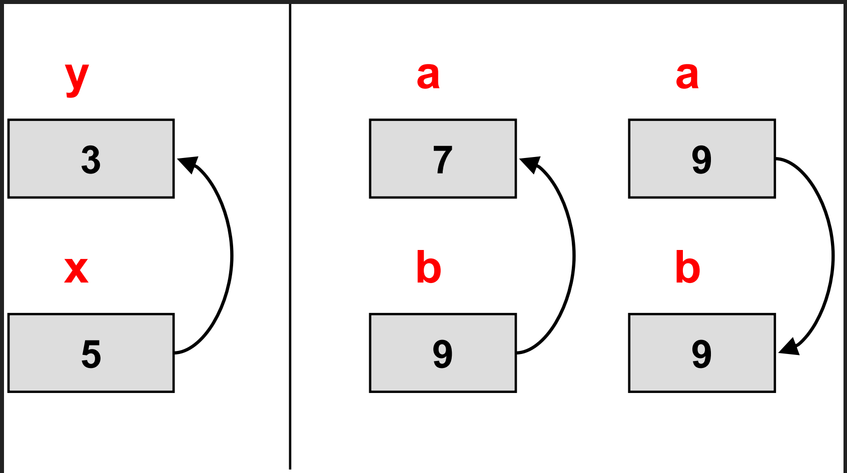
THINKING LIKE A C++ COMPILER

- Code is executed sequentially
  - You can assume each statement is executed, and finished before the next one starts

```
#include <iostream>

int main()

{

  int x = 10;

  int y;

  y = x/2;

  x = x + y;

  x /= 10;

}
```

PROGRAMMING CHALLENGE/EXERCISE

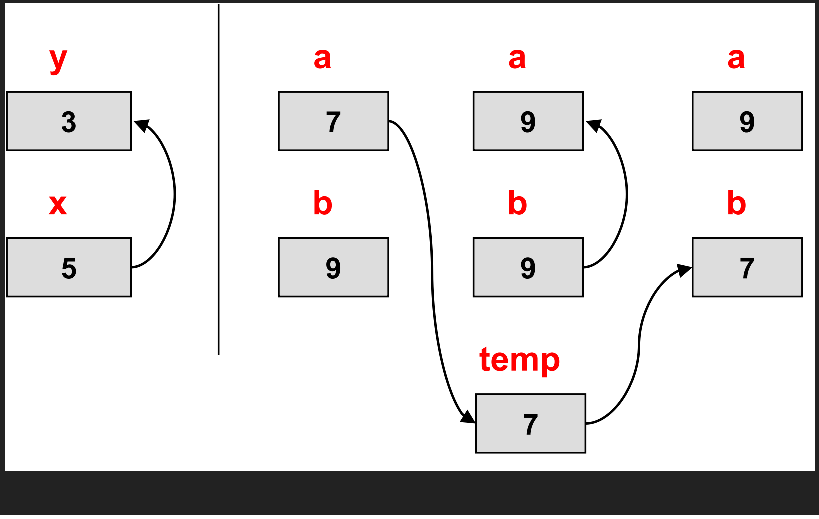
- How to swap the values of two variables?
- Will this work?
- In class exercise…



```
#include <iostream>

int main()

{

  int x = 5, y = 3;
  x=y; // copy y into x

  // now consider swapping

  // the value of 2 variables int
  a = 7;

  b = 9;
  a = b;
  b = a;

  cout << a << “ “ << b << endl;

}
```

PROGRAMMING CHALLENGE/EXERCISE

- How to swap the values of two variables?
- Swap requires temporary variable
  - We’ll come back to swap a few times

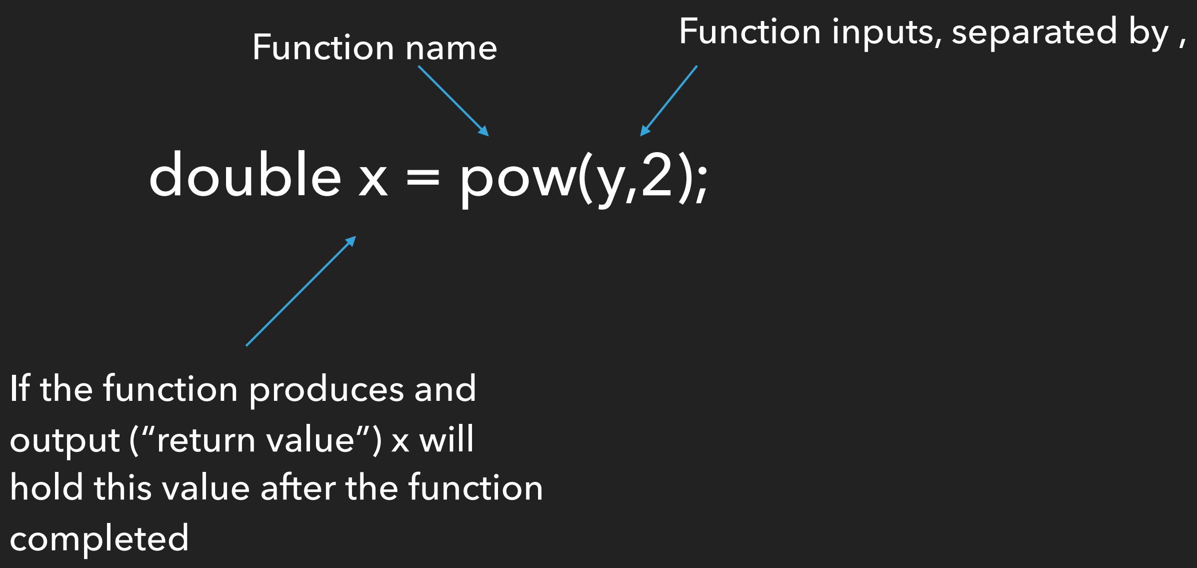


```
#include <iostream>

int main()

{

  int x = 5, y = 3;
  x = y; // copy y into x

  // let's try again
  int a = 7, b = 9, temp;

  temp = a;
  a = b;
  b = temp;


}
```

USING FUNCTIONS

- Functions are pieces of code, like mini-programs
  - They have a name and inputs
  - Usually produce outputs
- Lots of built-in functions you can use
- We’ll also write lots of functions

ANATOMY OF A FUNCTION CALL



Function name

Function inputs, separated by ,

```
double x = pow(y,2);
```

If the function produces and output (“return value”) x will hold this value after the function completed

BUILT-IN FUNCTIONS

- There are loads of built-in functions in C++ available with #include <>
  - sqrt(x): returns the square root of x (in <cmath>)
  - pow(x, y): returns xy, or x to the power y (in <cmath>)
  - sin(x): returns the sine of x if x is in radians (in <cmath>)
  - abs(x): returns the absolute value of x (in <cstdlib>)
  - max(x, y): returns the maximum of x and y (in <algorithm>)
  - min(x, y): returns the maximum of x and y (in <algorithm>)

BUILT-IN FUNCTIONS

```
#include <iostream>
#include <cmath>
#include <algorithm>

using namespace std;

int main(int argc, char *argv[]) {
    // can call functions
    //  in an assignment
    double res = cos(0);
    // can call functions in an
    // expression
    res = sqrt(2) + 2.3;
    // can call them as part of an output statement
    cout << max(34, 56) << endl;

    return 0;
}
```

MORE ON STATEMENTS

- Statements are basic building blocks of code
- End with ;
- Made up of
- assignments, arithmetic operators, function calls or a mix
  - sin(3.1415); //potential problem here
  - x++;
  - x = 5 + sin(x) - pow(y,2);

IN CLASS EXERCISES

- 4swap
- funccall
- hello

GETTING DATA INTO OR OUT OF OUR PROGRAMS

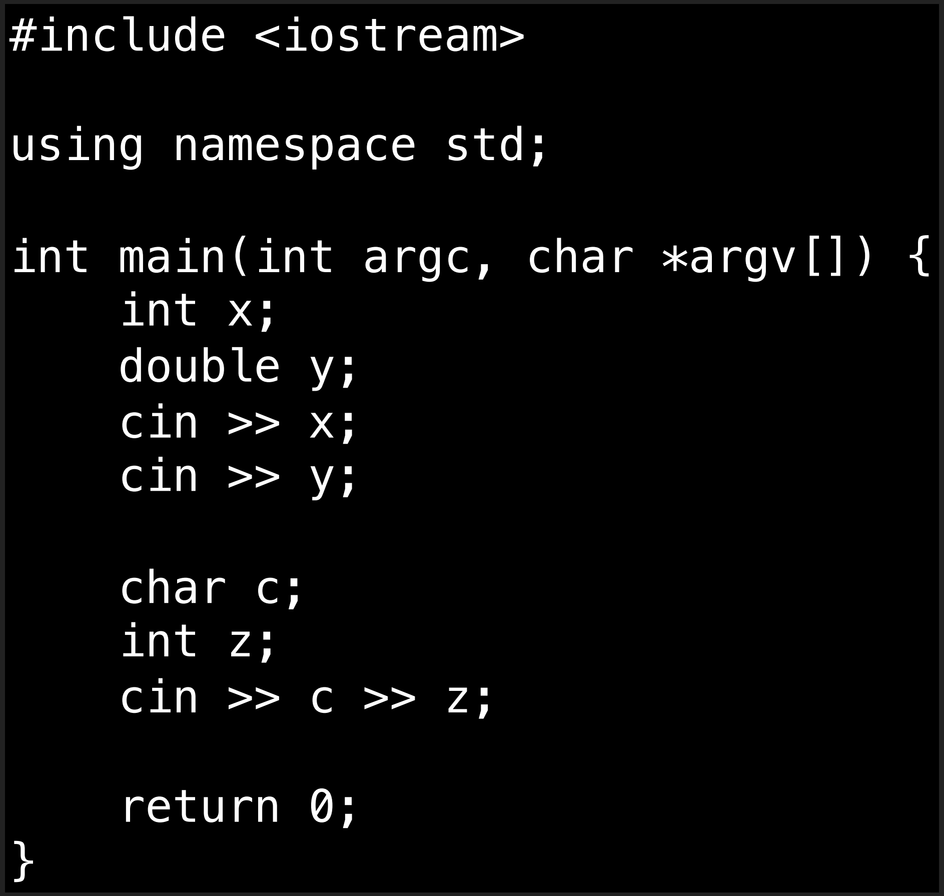
- C++ gives us an easy way to read from the keyboard and write to the terminal
- #include <iostream>
- using namespace std;
- cin (C standard input)
  - Read from the terminal (keyboard) into variables
- cout (C standard output) write formatted (interpreted) data to terminal

WHITESPACE

- Quick aside: whitespace
- Characters that we don’t “see”
  - newline, space, tab
- Comes up a lot over the semester

CIN

- For now reads from keyboard in your terminal
- skips (ignores) white space
- Use with >> (extraction operator)
- Reads characters and interprets into type of the variable on RHS
  - Can have more than one >> and variable in one statement
- If what you type can't be interpreted, silently "fails"



```
#include <iostream>

using namespace std;

int main(int argc, char *argv[]) {
    int x;
    double y;
    cin >> x;
    cin >> y;

    char c;
    int z;
    cin >> c >> z;

    return 0;
}
```

COUT

- Interprets data and writes to terminal
- Uses << (insertion) operator, can have more than one per statement;
- Use “endl;” to get a newline;

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[]) {
    int x = 10;
    double y = 2.5;
    cout << “x and y are:”;
    cout << x << “ “ << y << endl;

    return 0;
}
```

```
x and y are:10 2.5
```

IN CLASS EXERCISES

- tacos
- quadratic
- math

COMMENTS

/* anything between forward-slash-star and star-forward-slash are comments

including newlines

*/

or

// anything after double-forward-slashes is a comment until the next newline

PRE AND POST INCREMENT

- C++ has shortcut increment and decrement operators ++ --
- Position relative to variable matters
- x++; ++x
- y = x++ + --z;
- If the operator is before the variable, the variable is incremented (decremented) by one *before* the rest of the statement
- If the operator is after the variable, the statement is evaluated, then the variable is updated

PRE AND POST INCREMENT PRACTICE

- x = 3; int y;
- y = x++ + 5; (y = 8, x = 4)
- y = ++x + 5; (y = 9, x = 4)
- y = x– + 5; (y = 8, x = 2)

ACKNOWLEDGEMENTS

- All graphics from Wikimedia Commons unless otherwise noted
- Swap graphics and some examples courtesy Mark Redekopp